Reino Nikki

Mobiilipelin kehitys Unity-ympäristössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

2.12.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Reino Nikki Mobiilipelin kehitys Unity-ympäristössä 35 sivua + 1 liite 02.12.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Lehtori Jorma Räty

Työn tavoitteena oli tutkia Unity-pelinkehitysympäristöä ja sen soveltuvuutta pelien kehittämiseen. Tarkoituksena oli selvittää, mikä Unity on ja miten sen kanssa työskennellään. Työn ohessa kehitettiin mobiilipeli Android-käyttöjärjestelmälle Unity-ohjelmistoa käyttäen ja valmis peli julkaistiin Google Play -palvelussa.

Pelin kehityksen aikana huomattiin Unityn olevan helposti lähestyttävä, mutta silti erittäin monipuolinen pelinkehitysympäristö, joka tarjoaa erilaisen lähestymistavan pelinkehitysprosessiin ja avaa myös muille kuin ohjelmoijille oven pelikehityksen maailmaan.

Tutkimuksessa todettin, että Unity on maksuttoman lisenssinsä, monialustaisuutensa sekä yhteisöllisyytensä avulla nousemassa laajemman yleisön saataville, mutta ilmaislisenssin monet puutteet sekä maksullisten lisenssien suuri hinta pitävät vielä Unityn kaukana täydellisyydestä.

Avainsanat	Unity, Android, C#

Author(s) Title Number of Pages Date	Reino Nikki Developing Mobile Game in Unity Environment 35 pages + 1 appendix 2 December 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Jorma Räty, Senior Lecturer

This Thesis was made to study the game development environment Unity and how well it fares in developing games. The object was to find out what Unity is and how one should work with it. A mobile game for Android devices was developed with Unity as a part of the study and the completed game was published in the Google Play market.

During the development of the game, Unity showed remarkable diversity in its development toolset, while still maintaining an easily accessible interface into its rather original approach to game development. It seemed an original approach, which brings the world of game development closer to those unfamiliar with programming.

Although far from perfect due to the high prices of its licenses and the shortcomings of its free license, Unity's free license, multiple-platform support and community make it a viable option for wider audiences.

Keywords	Unity, Android, C#

Sisällys

Lyhenteet ja käsitteet

GDD	Game Design Document. Pelinsuunnitteludokumentti, johon tehdään pelin määrittely.
C#	on Microsoft-yhtiön .NET-konseptia varten kehittämä ohjelmointikieli.
Mesh	Polygoniverkko (engl. Polygon mesh) eli kokoelma geometrisiä kärkiä, särmiä ja tahkoja, jotka muodostavat kappaleita.
Prefab	Unityn käyttämä etukäteen rakennettu objekti (engl. Prefabricated object).

1 Johdanto

Toukokuussa 2013 Unity-pelinkehitysympäristön kehittäjäyhtiö, Unity Technologies, ilmoitti, että aikaisemmin maksulliset Android- ja iOS-lisenssit kuuluvat jatkossa Unityn ilmaislisenssiin [1]. Lisenssi mahdollistaisi näille alustoille tarkoitettujen pelien kehittämisen Unityllä. Tuohon aikaan olin harjoittelussa pienessä suomalaisessa mobiilipeliyhtiössä, Mobilivessä. Yhtiössä käytettiin pelinkehitykseen Unityä, jonka käytön jouduin omaksumaan itsekin. Kuinka ollakaan, ihastuin Unityyn täydellisesti.

Valitsin insinöörityöni aiheeksi ”Mobiilipelin kehitys Unity-ympäristössä”, koska aiemmin mainitun uutisen vuoksi Unity on nyt ajankohtaisempi kuin koskaan. Uutinen koskettaa etenkin indie-pelinkehittäjiä, ja uskon, että pystyn laskemaan mobiilipelin kehittämisen aloituskynnystä tämän työn avulla.

Työn aiheena on kehittää mobiilipeli Android-käyttöjärjestelmälle ja julkaista se Google Play -palvelussa. Pelin kehitykseen käytetään Unity-ohjelmistoa. Kehitysprosessin yhteydessä tutkitaan Unityn, etenkin Unityn ilmaisversion, soveltuvuutta pelien kehittämiseen. Lisäksi pohditaan kehitystyötä helpottavia työtapoja sekä perehdytään Google Play -palvelun ominaisuuksiin.

Ensimmäiseksi kerrotaan käytetystä pelinkehitysympäristöstä, Unitystä. Tutkitaan Unityn toimintaperiaatteita sekä teknologiaa, jonka päälle se on rakennettu. Lisäksi käydään läpi pelinkehittäjille olennaisia asioita, kuten Unityn käytön hintaa sekä sen kanssa työskentelyä.

Kolmannessa luvussa kerrotaan kehitettävästä pelistä ja sen kehitystyöstä, jonka jälkeen selvitetään Google Play -palvelun mysteereitä sekä pelin julkaisuprosessia. Lopuksi viidennessä luvussa pohditaan Unityn soveltuvuutta tämän projektin toteutukseeen sekä mobiilipelinkehittämiseen ja pelinkehittämiseen yleensä.

2 Unity

2.1 Yleistä

Unity on alustariippumaton pelinkehitysympäristö, jota alettiin kehittää 2001. Ensimmäinen Unity-versio julkaistiin 2005 [2]. Unityn yhtenä tarkoituksena on minimoida pelinkehittäjän matalan tason ohjelmoinnin tarve. Toisin sanoin Unityn tarkoitus on tehdä pelinkehityksestä helppoa ja antaa mahdollisuus pelien tekemiseen niille, jotka eivät ymmärrä pelien alla toimivaa ohjelmistotekniikkaa eivätkä siksi muuten pystyisi siihen. Unity pyrkii tarjoamaan kaikki pelinkehitykseen tarvittavat työkalut yhdessä paketissa. Omilla verkkosivuillaan Unityä luonnehditaankin pelinkehitysekosysteemiksi, joka sisältää edistyksellisen, voimakkaan grafiikka- ja fysiikkamoottorin sekä ohjelmointiympäristön.

2.1.1 Monialustaisuus

Yksi Unityn suurista valttikorteista on sen monialustaisuus. Eri alustoille julkaiseminen tapahtuu helposti, sillä Unity tuottaa itse alustakohtaiset ohjelmakoodit. Unity tukee kehitystä seuraaville alustoille:

- iOS
- Android
- Windows
- BlackBerry 10
- Mac
- Linux
- nettiselaimet
- PlayStation 3
- Xbox 360
- Windows Phone 8
- Wii U.

Alustoille julkaiseminen vaatii alustakohtaisen lisenssin. PlayStation 3-, Xbox 360-, Wii U- ja Wii-alustojen lisenssit neuvotellaan tapauskohtaisesti, mutta muut lisenssit ovat ostettavissa Unityn Asset Storesta tai kuuluvat valmiiksi Unityyn. Unity on aikaisemmin tukenut myös Flash-alustoille, mutta huhtikuussa Unity Technologies ilmoitti lopettavansa Flash-lisenssien tarjoamisen. [3.]

2.1.2 Scene-tiedostot

Unity-projekti koostuu yhdestä tai useammasta scene-tiedostosta. Scene-tiedostot ovat ”kohtauksia”, jotka sisältävät tiedon siitä, mitä peliobjekteja pelissä on, missä ne ovat, mitä ne tekevät, minkä kokoisia ne ovat ja niin edelleen. Useimmissa Unity-projekteissa yksi scene-tiedosto vastaa yhtä pelin tasoa, huonetta tai muunlaista yhtä kokonaisuutta.

2.1.3 Assetit

Assetit ovat kaikki projektiin liittyvä sisältö, kuten 3D-mallit, kuvatiedostot, äänitiedostot ja skriptit. Tarkemmin sanottuna assetteja ovat näistä lähdetiedostoista tuodut tiedostot, jotka ovat Unityn käyttämässä tiedostomuodossa, mutta käyttäjälle nämä ovat käytännössä samoja asioita. Lähdetiedostojen tuotujen versioiden tuomisasetuksia, esimerkiksi kuva- tai äänitiedostojen pakkausasetuksia, voidaan muokata Unityn käyttöliittymässä.

2.1.4 Peliobjektit

Peliobjektit (engl. Game Object) koostuvat yhdestä tai useammasta komponentista, joista yksi on aina transform-komponentti. Tämä komponentti kertoo objektin position, rotaation ja skaalan. Transform-komponentin lisäksi peliobjekti sisältää yleensä muita komponentteja, kuten skriptejä tai meshejä, jotka antavat peliobjektille lisää määrityksiä. Usein peliobjekti on jokin kappale, kuten kuutio, pallo tai jokin monimutkaisempi malli, kuten ihminen. Peliobjekti voi kuitenkin olla myös ”tyhjä”, eli sillä voi olla transform-komponentti ja skriptejä, mutta se ei näy pelinäkymässä. Tällaisia käytetään muun muassa silloin, kun halutaan asettaa peliin skripti, mutta ei haluta lisätä sitä jo olemassa olevaan objektiin.

Peliobjektit muodostavat keskenään lapsi-vanhempi-hierarkian. Yksi peliobjekti voidaan asettaa toisen peliobjektin lapseksi, jolloin sen transform-komponentin tiedot perustuvat sen vanhemman transformiin. Myös kaikki vanhemman transformiin tehtävät muokkaukset näkyvät lapsen transformiin. Toisin sanoin lapsiobjekti liikkuu, pyörii ja skaalautuu vanhempansa mukana.

2.1.5 Prefab-tiedostot

Luotu peliobjekti voidaan tallentaa prefab-tiedostoksi, joka toimii eräänlaisena template-objektina. Näitä voidaan luoda useita instansseja. Ne säilyttävät yhteyden prefabiin. Prefabiin ja sen komponentteihin tehdyt määritykset ja muutokset siirtyvät automaattisesti myös kaikkiin sen instansseihin, mikäli niitä ei ole ylikirjoitettu instanssissa. Prefabeista on hyötyä tilanteissa, joissa samanlaisia peliobjekteja on paljon, jolloin käyttäjän tarvitsee tehdä vain yksi prefabi, josta luodaan instanssi sinne, missä sitä tarvitaan. Prefabit mahdollistavat myös peliobjektien instanssien luomisen ajon aikana.

2.2 Lisensointi

2.2.1 Unity Free

Unity Free on Unityn maksuton versio, ja se on saatavilla yksityisille pelintekijöille sekä pienille kaupallisille yrityksille, joiden vuotuinen liikevaihto on alle 100 000 dollaria. Lisenssin omistaja ei myöskään saa olla opetuslaitos, jonka vuotuinen kokonaisbudjetti ylittää 100 000 dollaria. Maksuton lisenssi rajoittaa joidenkin Unityn ominaisuuksien käyttöä. Lisäksi maksuttomalla versiolla tuotettuun peliin lisätään Unityn logon sisältävä splash screen, jota ei voi poistaa tai muokata. Maksuttomalla versiolla tuotettuihin web-peleihin lisätään splash screenin sijaan Unityn logon sisältävä vesileima. [4.]

2.2.2 Unity Pro

Unity Pro on Unityn maksullinen versio, ja se on tarkoitettu suuremmille pelintekijöille tai pelifirmoille. Unity Pro -lisenssi lisää maksuttomaan versioon edistyksellisempiä ominaisuuksia, kuten katve-erottelun.

Taulukko 1. Unityn ilmaisversion ja maksullisen version ominaisuuksien vertailu. [Lähde: http://unity3d.com/unity/licenses]

Ominaisuus	**Unity Free**	**Unity Pro**
Physics	x	x
NavMeshes, path-finding, and crowd Simulation	x	x
Multiplayer Networking with RakNet	x	x
LOD-support		x
Audio (3D Positional and Classic Stereo)	x	x
Audio Filter		x
Video Playback and Streaming		x
Mecanim	x	x
Mecanim: IK Rigs		x
Mecanim: Sync Layers & Additional Curves		x
One-Click Deployment	x	x
Web Browser Integration	x	x
Custom Splash Screen		x
Build Size Stripping		
Low-Level Rendering Access	x	x
Dynamic Fonts with markup	x	x
Shuriken Particle System	x	x
3D Texture Support		x
Realtime Directional Shadows	x	x
Realtime Spot/Point and soft shadows		x
HDR, tone mapping		x
Light Probes		x
Optimized Graphics	x	x
Shaders (Built-in and Custom)	x	x
Lightmapping	x	x
Lightmapping with Global Illumination and area lights		x
Dynamic Batching	x	x
Static Batching		x
Terrains (Vast, Densely Foliaged Landscapes)	x	x
Render-to-Texture Effects		x
Full-Screen Post-Processing Effects		x
Occlusion Culling		x
Deferred Rendering		x
Full multi-screen support (AirPlay)		
Stencil Buffer Access		x
GPU Skinning		x
Navmesh: Dynamic Obstacles and Priority		x
Webplayer debugging	x	x
.NET Based Scripting with c#, JavaScript and Boo	x	x
Access to Web Data through WWW Functions	x	x

Open an URL in the User's Browser	x	x
.NET Socket Support	x	x
Native Code Plugins Support		x
Inspector GUI for custom classes	x	x
Integrated Editor	x	x
Instantaneous, Automatic Asset Importing	x	x
Profiler and GPU profiling		x
External Version Control Support	x	x
Script Access to Asset Pipeline		x
Dark Skin		x

Unity Pro -lisenssin voi ostaa noin 1 500 dollarin kertahinnalla. Lisäksi se on saatavilla myös 75 dollarin kuukausimaksulla, mikä vaatii vähintään 12 kuukauden tilauksen. [5.]

2.2.3 Alustakohtaiset lisenssit

Unityllä on mahdollista kehittää pelejä useille eri alustoille. Julkaiseminen tietyllä alustalla vaatii alustakohtaisen lisenssin. Toukokuussa 2013 Unity Technologies ilmoitti, että alun perin maksullisia olleet Unityn Android-, iOS-, Windows Phone 8 sekä BlackBerry 10 kuuluvat jatkossa Unityn ilmaislisenssiin [6].

Unity Pro -lisenssin omistajien on mahdollista hankkia alustakohtaisia Pro-tason lisenssejä. Windows Phone 8 Pro ja Windows Store Apps Pro -lisenssit kuuluvat Unity Pro -lisenssiin, mutta muut ovat saatavilla 1 500 dollarin kertahintaan tai 75 dollarin kuukausitilauksella. [5.]

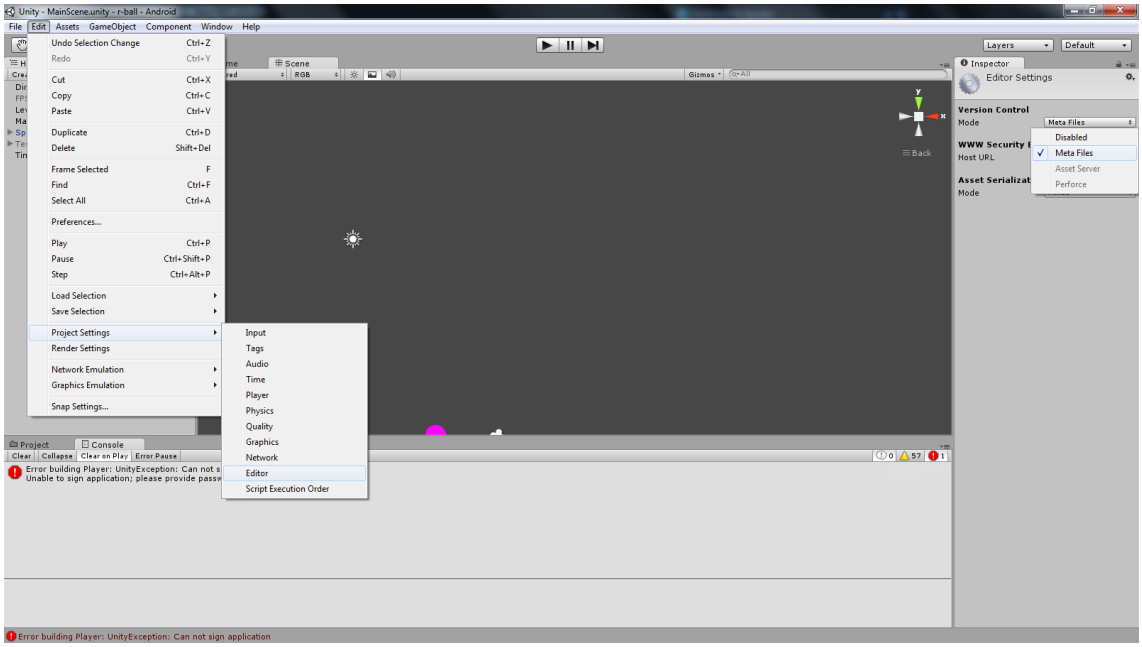
2.2.4 Ryhmätyöskentely ja versionhallinta

Unity tarjoaa ryhmätyöskentelyn helpottamiseksi 500 dollarin hintaista Team License -lisenssiä. Lisenssi antaa käyttäjille mahdollisuuden käyttää versionhallintaan ulkoista versionhallintaohjelmaa tai Unityn Asset Server -versionhallintaohjelmaa. Unity Asset Server on täysin Unitya varten suunniteltu versionhallintaohjelma. [5.]

Oman versiohallinnan lisäksi lisenssi mahdollistaa Unity Asset Cache Server -palvelun käytön. Palvelun pitäisi nopeuttaa muokattujen lähdetiedostojen projektiin uudelleentuomista siten, että tiedoston tuotu data tallennetaan välimuistiin palvelimelle.

Kun käyttäjät ottavat versionhallinnasta muutetun projektin, muutettuja lähdetiedostoja ei tarvitse enää tuoda uudelleen projektiin, vaan tuotu data otetaan suoraan palvelimen välimuistista. Erittäin suurissa projekteissa lähdetiedostojen tuominen projektiin saattaa kestää jopa tunteja, joten Unity Asset Cache Server on tarpeellinen suurissa ryhmätyöprojekteissa. [7.]

Unity tukee ulkoisen versionhallintaohjelman käyttöä. Ulkoisen versionhallinnan käyttö edellyttää meta-tiedostojen käyttöönottoa (kuva 1). Jokaiselle projektin tiedostolle luodaan meta-tiedosto, jonka avulla muun muassa assettien asetukset saadaan siirtymään projektista toiseen. Käyttäjän on pidettävä huoli, että versionhallintaan lisätään normaalien tiedostojen lisäksi kaikki meta-tiedostot. [8.]



Kuva 1. Meta-tiedostojen aktivoiminen Unityn editorissa.

Jos projekti on suuri ja sisältää paljon assetteja, on myös metatiedostojen määrä suuri. Tämä kasvattaa projektin kokoa ja voi hankaloittaa projektin kanssa työskentelyä.

Toinen ongelma esiintyy scene-tiedostojen ja prefabien versioimisen yhteydessä. Nämä tiedostot ovat binääri-tiedostoja, joten tiedostoihin tehtyjen muutosten yhdistäminen eli niiden mergeäminen ei ole mahdollista versionhallintaohjelmissa. Tämä tarkoittaa käytännössä sitä, että useampi kuin yksi käyttäjä ei voi muokata näitä tiedostoja samanaikaisesti. Ongelman voi välttää asettamalla versionhallintaohjelmasta

näille tiedostoille pakolliset lukot, jotka estävät usean käyttäjän samanaikaisen muokkauksen. Tämä estää ongelman esiintymisen, mutta ei poista sitä.

Unity Pro -lisenssin omistajien on mahdollista asettaa assettien sarjallistaminen tekstimuotoiseksi. Tämä tarkoittaa sitä, että versionhallintaan lisättäessä Unity muuntaa binääritiedostot tekstimuotoon. Tekstimuotoisina niiden mergeäminen onnistuu, ja useampi käyttäjä voi muokata tiedostoja samanaikaisesti ilman ongelmia. [9]

2.3 Pelimoottori

2.3.1 Renderöinti

Unityn grafiikkamoottori käyttää Direct3D-ohjelmointirajapintaa Windows- ja Xbox 360 -alustoille julkaistaessa ja OpenGL-rajapintaa kun alustana toimii Mac, Windows, Linux tai PlayStation 3. Androidille ja iOS:lle julkaistaessa käytetään OpenGL ES -rajapintaa.

Unity tukee lukuisia ulkopuolisilla graafikka-ohjelmilla tuotettuja tiedostoja. Tuettuja ohjelmia ovat ainakin

- 3ds
- Max
- Maya
- Softimage
- Blender
- modo
- ZBrush
- Cinema 4D
- Cheetah3D
- Adobe Photoshop
- Adobe Fireworks.

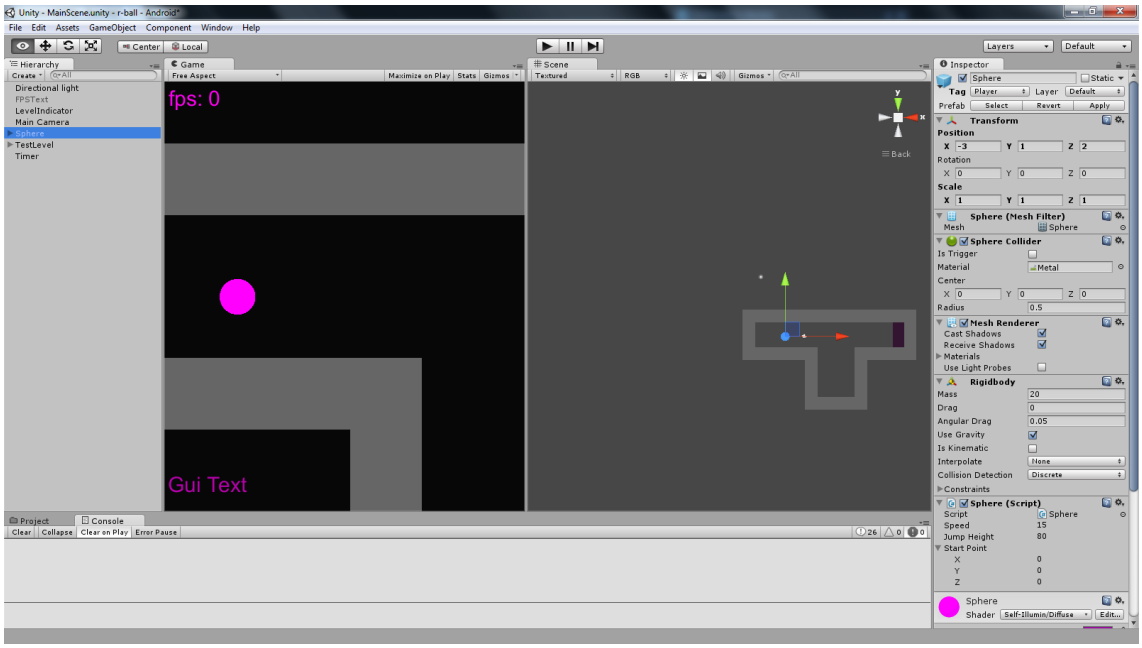
Tiedostojen käyttäminen Unityssä on yksinkertaista. Tiedostot siirretään projektin kansiorakenteeseen, esimerkiksi Windowsin resurssinhallinnan kautta, jolloin Unity tuo ne automaattisesti projektiin. Tämän jälkeen niitä on mahdollista hallita Unityn oman graafisen käyttöliittymän kautta. [10.]

2.3.2 Fysiikanmallinnus

Unityssä on sisäänrakennettu fysiikanmallinnus, joka on toteutettu Nvidian PhysX -fysiikkamoottorilla. Moottori on monipuolinen ja se tukee useita eri fysiikkamallinnuksia, kuten kangasmallinnus, jäykät ja pehmeät, erilaisten nivelien ja jousien mallinnus sekä räsynukkemallinnus. [11] Fysiikkamoottori ottaa mallinnuksessa automaattisesti huomioon kaikki peliobjektit, joilla on komponenttina jokin fysikkakomponentti, esimerkiksi RigidBody-komponentti.

2.4 Editori

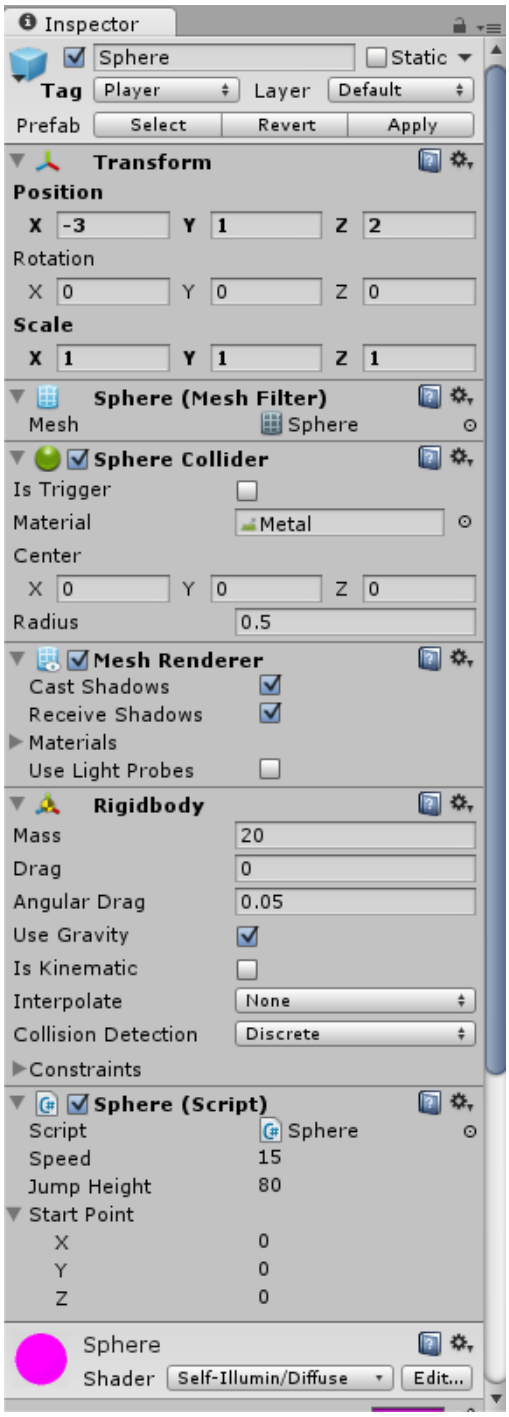
Unityn editori on Unityn perusnäkymä, josta käyttäjä voi tutkia aukinaista sceneä sekä hallinnoida assetteja. Editori on ohjelmointiympäristön rinnalla yksi Unityn kahdesta päänäkymästä. Editorin näkymä koostuu useista ikkunoista. Käyttäjä voi valita näkyviin haluamansa ikkunat ja järjestellä niitä vapaasti. Yleensä näkyvissä ovat kuitenkin hierarkiaikkuna, peli-ikkuna, scene-ikkuna, projekti-ikkuna, inspector-ikkuna sekä projekti-ikkuna (kuva 2).



Kuva 2. Yksi mahdollinen perusnäkymän asetelma Unityssä.

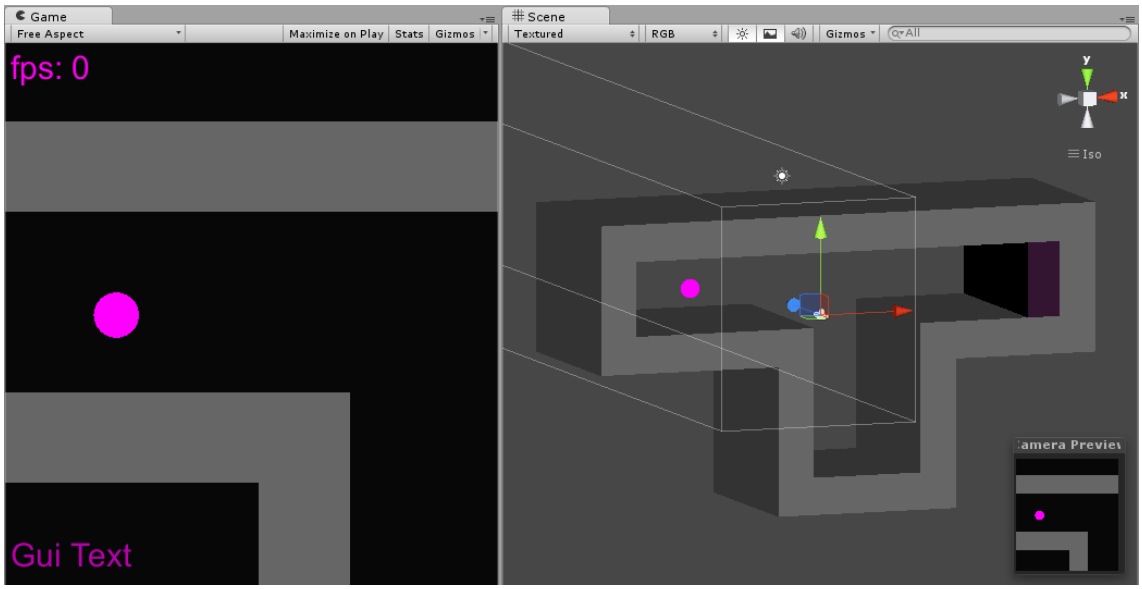
Hierarkia-ikkuna näyttää listan scenellä olevista peliobjekteista, sekä niiden välisen hierarkian. Tässä ikkunassa käyttäjä voi luoda uusia peliobjekteja, valita ja nimetä jo olemassa olevia peliobjekteja tai muunnella peliobjektien välistä hierarkiaa.

Inspector-ikkunassa (kuva 3) näkyy lista valitun peliobjektin komponenteista sekä komponenttien muuttujista. Skriptin kaikki public-määreellä varustetut muuttujat ovat näkyvissä ja muokattavissa inspectorissa skriptin alla. Arvot ovat näkyvissä ja muokattavissa myös silloin, kun peli on käynnissä. Tämän ansiosta ohjelmoijan on helppo seurata muuttujien arvoja suorituksen aikana, mikä helpottaa muun muassa testausta ja debuggausta.



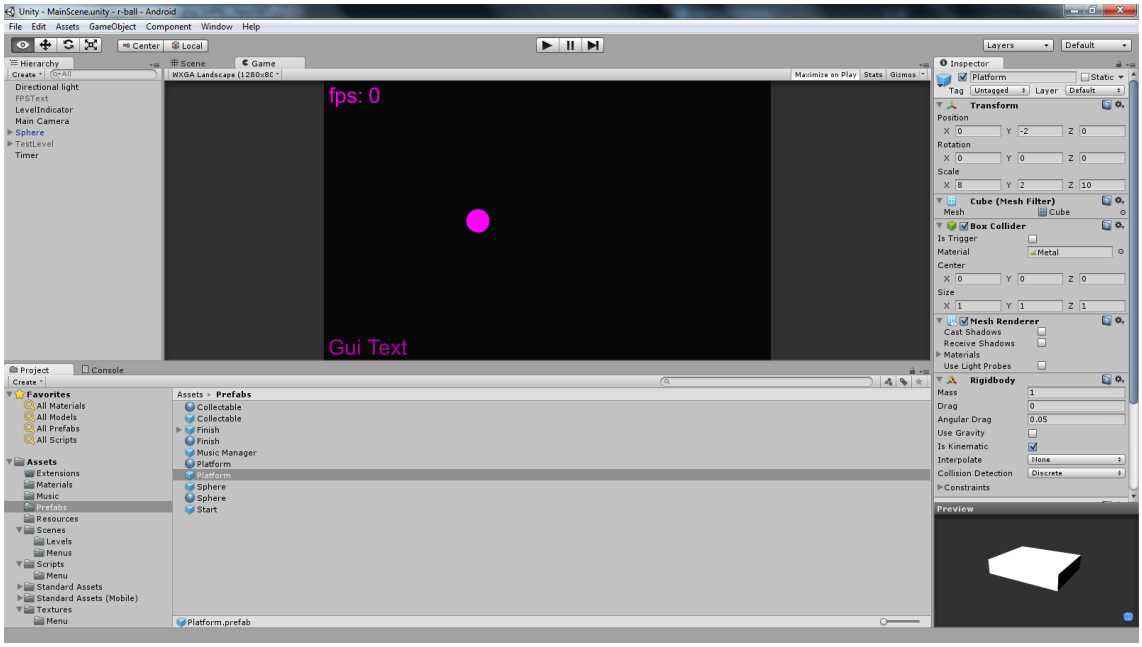
Kuva 3. Inspector-ikkuna

Scene-ikkunassa (kuva 4) näkyvät scenelle lisätyt peliobjektit. Tässä ikkunassa käyttäjä voi siirrellä ja muokata vapaasti peliobjekteja.



Kuva 4. Peli-ikkuna ja scene-ikkuna.

Peli-ikkuna näyttää, mitä pelin kamera näkee. Käyttäjä voi play-nappia painamalla käynnistää pelin. Käynnissä oleva peli tulee näkyviin peli-ikkunaan sitä, mitä nähtäisiin valmiissa pelissä. Käynnissä olevaa peliä voi tarkastella samanaikaisesti sekä peli-ikkunassa että scene-ikkunassa. Käyttäjä voi jopa suoraan hallinnoida käynnissä olevan scenen peliobjekteja.



Kuva 5. Projekti-ikkuna ja valitun assetin näkyminen inspectorissa.

Projekti-ikkunasta (kuva 5) käyttäjä näkee projektin kansiorakenteen, joka vastaa käyttöjärjestelmän vastaavaa. Käyttäjä voi tässä ikkunassa hallita ja luoda assetteja, lisätä scenelle prefabeja tai avata toisen scene-tiedoston. Käyttäjä voi tarkastella valittuja assetteja inspectorissa.

2.5 Asset Store

Marraskuussa 2010 käynnistettiin Unity Asset Store (kuva 6), Unityn Editorin sisäinen asset-kauppa [12]. Kaupassa on suuri määrä asset-paketteja, kuten 3D-malleja, tekstuureja, materiaaleja sekä laajennuksia editoriin. Lisäksi kaupassa tarjotaan aloittelijaystävällistä sisältöä, kuten harjoittelu- ja esimerkkiprojekteja sekä tutoriaaleja.

Kuva 6. Asset Store Unityn editorissa.

Asset Storessa on sekä maksullista että maksutonta sisältöä.

2.6 Ohjelmointiympäristö ja skriptit

Unityn skriptit on toteutettu Mono 2.6 -kehitysympäristön ympärille, joka pohjautuu .NET Frameworkiin. Unityn mukana tulee mukautettu versio MonoDevelopista, joka on

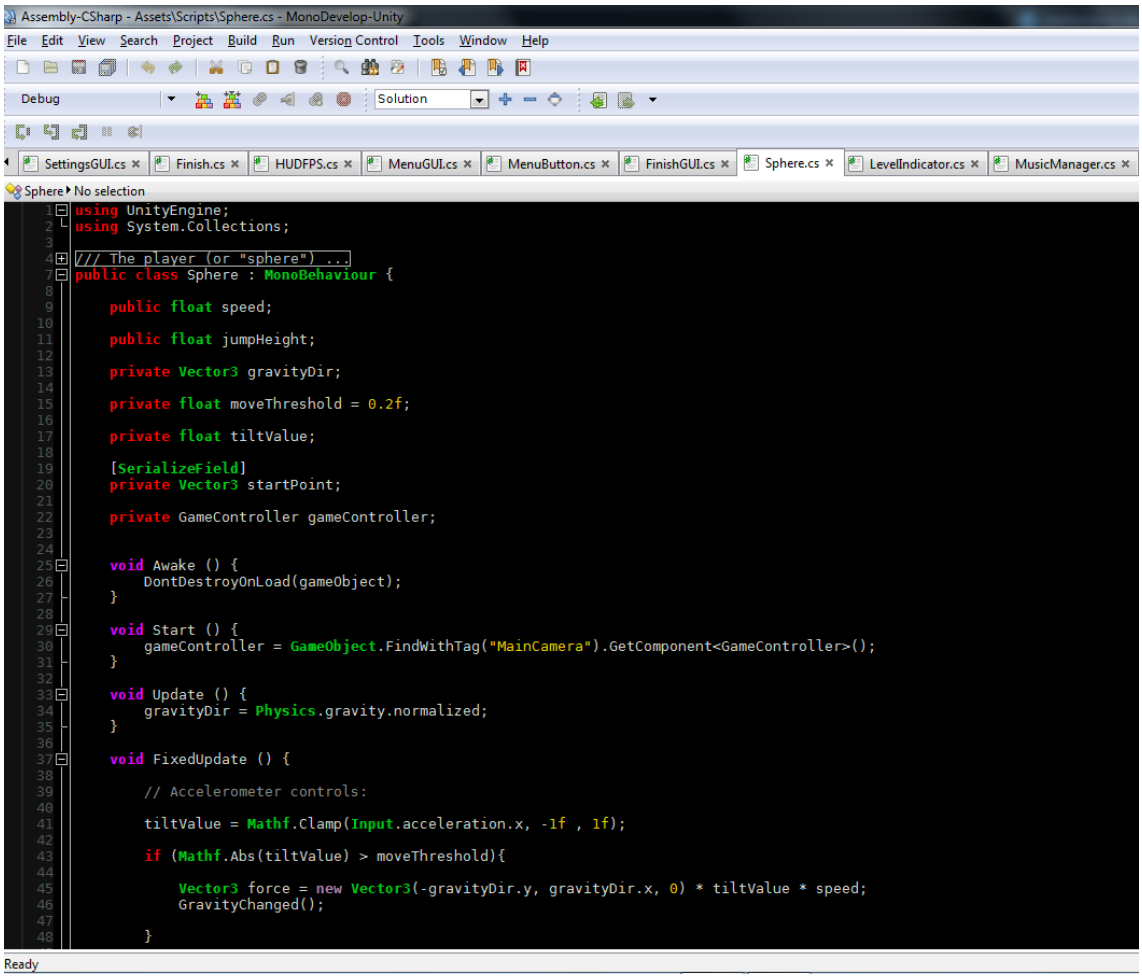
avoimen lähdekoodin ohjelmointiympäristö. MonoDevelopissa on monia muista ympäristöistä tuttuja ominaisuuksia, kuten tekstintäydennys, ja versionhallinta. [10]

Unity tukee kolmen eri ohjelmointikielen käyttöä. Käytettäviä kieliä ovat C#, Boo ja JavaScript, joista C# ja JavaScript ovat selvästi eniten käytettyjä. Unityssä käytettävä JavaScript on Unityn oma muunnelma ECMAScript-standardin JavaScript-kielestä, ja siksi sitä kutsutaankin usein UnityScriptiksi.

Unityssä kaikki ohjelmakoodit ovat skriptejä, joita on kahdenlaisia: peliobjektin toimintaa ohjaavia MonoBehaviour-skriptejä sekä editoria muokkaavia editori-skriptejä.

2.6.1 MonoBehaviour-skriptit

MonoBehaviourit ovat skriptejä, joiden luokka periytyy MonoBehaviour-luokasta (kuva 7). Jokaisen MonoBehaviour-skriptin tulee toimiakseen olla jonkin peliobjektin komponenttina.



Kuva 7. MonoDevelopissa kirjotettu MonoBehaviour-skripti

Unity ei oletusarvoisesti tue usean säikeen samanaiakaista ajoa, vaan kaikki ajetaan yhdessä pääsäikeessä. Siksi koodin ajaminen on toteutettu Unityssä hieman normaalista poiketen. Skriptejä ajettaessa ei käytetä yhtä entry-pistettä. Toisin sanoin koodin suoritus ei ala tietystä kohdasta, vaan skriptejä ajetaan ”samanaikaisesti”. Ohjelmoija voi hieman vaikuttaa skriptien suorituksprioriteetteihin, mutta muuten skriptit suoritetaan ohjelmoijalle tuntemattomassa järjestyksessä, joka saattaa myös muuttua.

Koodi ajetaan osioittain siten, että tiettyjä metodeja kutsutaan aina tietyssä järjestyksessä. Ensin suoritetaan yksi osio jokaisesta skriptistä, sitten toinen osio jokaisesta skriptistä, ja niin edelleen.

Taulukko 2. Koodin ajoa rytmittävät tapahtuma-metodit.

Metodin nimi	**Kuvaus**
Awake	Kutsutaan ennen Start-metodeja ja välittömästi prefabin instantioimisen jälkeen. Ei-aktiivisten peliobjektien Awake-kutsut tapahtuvat vasta objektin aktivoimisen jälkeen.
OnEnable	Kun MonoBehaviour-instanssi luodaan. Esimerkiksi silloin, kun ladataan uusi scene.
Start	Kutsutaan ennen ensimmäistä ruudunpäivitystä.
OnApplicationPause	Sen framen jälkeen, jossa tauko (pause) huomataan. Kutsun jälkeen näytetään yksi ylimääräinen frame tauko-grafiikoiden näyttämiseksi.
FixedUpdate	Kiintein aikavälein. FixedUpdate on frame ratesta riippumaton, sillä se toimii kiinteällä aikavälillä.
Update	Kerran framessa.
LateUpdate	Kerran framessa, Updaten suorituksen jälkeen. Kaikki Updaten laskutoimitukset on suoritettu loppuun ennen kuin LateUpdatea kutsutaan.
OnPreCull	Kutsutaan ennen kuink kamera erottelee scenen.
OnBecameVisible	Kutsutaan, kun objekti tulee näkyviin mihin tahansa kameraan.
OnBecameInvisible	Kutsutaan, kun objekti poistuu minkä tahansa kameran näkymästä.
OnWillRenderObject	Kutstutaan kerran jokaisessa kamerassa, jos objekti on näkyvissä.
OnPreRender	Kutsutaan ennen kuin kamera aloittaa scenen renderöinnin.
OnRenderObject	Kutsutaan, kun kaikki tavallinen scenen renderöinti on suoritettu.
OnPostRender	Kutsutaan, kun kamera on lopettanut scenen renderöinnin.
OnRenderImage (Vain Pro-lisensseillä)	Kutsutaan, kun scenen renderöinti on valmis ja mahdollistetaan näytön kuvan jälkikäsittely.
OnGui	Kutsutaan useita kertoja framessa GUI-tapahtumien mukaisesti.
OnDrawGizmos	Käytetään Gizmojen piirtämiseen scene-näkymään visualisointitarkoituksiin.
OnDestroy	Kutsutaan, objektin olemassaolon viimeisen framen kaikki frame updatet on suoritettu.
OnApplicationQuit	Kutsutaan kaikkissa peliobjekteissa ennen kuin sovellus suljetaan. Editorissa sitä kutsutaan, kun käyttäjä pysäyttää peli-tilan.
OnDisable	Kutsutaan kun scriptistä tulee ei-aktiivinen tai se disabloitu.

Skriptien koodin ajaminen suoritetaan järjestyksessä:

- kaikki Awake-kutsut.
- kaikki Start-kutsut
- kun lähestytään delta time -muuttujaa:
  - Kaikki FixedUpdate-kutsut
  - fysiikan simulointi

  - OnEnter/Exit/Stay triggeritapahtumakutsut
  - OnEnter/Exit/Stay törmäystapahtuma
- rigidbodyen transformien position ja rotaation interpolointi
- syötetapahtumat, kuten nappienpainallukset ja ylöspäästämiset
- kaikkien Update-metodien läpikäyminen
- animaatioiden eteneminen
- kaikkien LateUpdate-metodien läpikäyminen
- renderöinti

Skriptin omien muuttujien initialisointi kannattaa tehdä Awake-metodin sisällä ja mahdolliset viittaukset toisiin objekteihin Start-metodin sisällä, koska kaikkien skriptien Awake-metodit käydään läpi ennen ensimmäistäkään Start-metodia. Näin kaikki skriptien ja objektien väliset viittaukset löytävät jo initialisoituja arvoja.

Unityssä on kolme paljon käytettyä metodia, joita iteroidaan ohjelman ajon aikana. Näissä metodeissa voi tehdä iteraatioita, kuten animaatioita ja kameran siirtämistä: FixedUpdate-, Update- ja LateUpdate-metodit.

Update-metodi on eräänlainen Unityn ”työjuhta”, sillä se sisältää usein suurimman osan MonoBehaviourin toiminnoista ja laskutoimituksista. Updatea kutsutaan kerran jokaisen framen aikana, joten se on hyvä paikka monenlaisten käyttäytymisten toteuttamiseen.

FixedUpdate-metodia kutsutaan kiinteän aikavälin mukaan, toisin kuin Update-metodia. Kaikki fysiikkamoottorin suorittamat laskutoimitukset ja päivitykset tapahtuvat välittömästi FixedUpdaten suorituksen jälkeen. Tämän vuoksi on suositeltavaa tehdä kaikki skriptin fysiikanmallinnustoiminnot tämän metodin sisällä. Koska FixedUpdate toimii frame ratesta riippumattomasti, ei sen sisällä liikettä laskiessa tarvitse myöskään selvittää edellisen framen kestoa.

LateUpdate-metodia kutsutaan kerran jokaisen framen aikana, mutta vasta kun kaikki Update-metodit on suoritettu loppuun. Tämä on hyvä paikka operaatioille, jotka käyttävät sellaista tietoa, mikä saattaa muuttua Updaten aikana. Näin siksi, että eri Update-metodien kutsumisjärjestystä ei ole määritelty, ja käytetty tieto saattaa olla

vanhentunut framen aikana. Esimerkiksi peliobjektia seuraava kamera on hyvä toteuttaa täällä, sillä peliobjekti saattaa liikkua oman Updatensa sisällä. [13.]

2.6.2 MonoBehaviour-skriptien välinen kommunikaatio

Koska skriptit toimivat peliobjektien komponentteina, yhdestä skriptistä voi olla samanaikaisesti useita instansseja. Jos skriptistä halutaan päästä käskiksi toisen skriptin muuttujiin tai metodeihin, tähän on Unityssä kolme tapaa.

Skriptissä voi olla suora viittaus johonkin kohdeskriptin instanssiin. Viittaus saadaan etsimällä sceneltä oikea peliobjekti ja hakemalla sen komponenteista haluttu skripti.

```
22      private GameController gameController;
23
24
25      void Awake () {
26          DontDestroyOnLoad(gameObject);
27      }
28
29      void Start () {
30          gameController = GameObject.FindWithTag("MainCamera").GetComponent<GameController>();
31      }
```

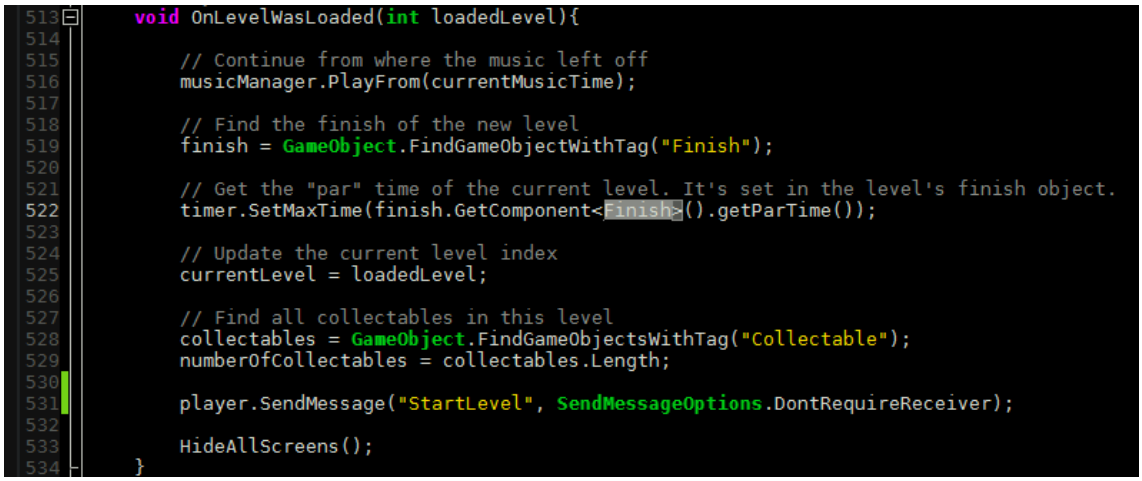
Kuva 8. Esimerkki peliobjektin etsimisestä ja peliobjektin komponentin noutamisesta.

Peliobjektin etsimisprosessissa (kuva 8) voi käyttää peliobjektin nimeä tai tagiä. Tagit ovat lista tunnistimia, joista osa on Unityn ja osa käyttäjän määrittelemiä. Peliobjektin voi merkitä tagilla, jolloin sen etsiminen helpottuu.

Etsimisprosessit voivat olla hitaita, etenkin jos peliobjekteja on runsaasti, ja niiden kirjoittaminen ohjelmakoodiin vie aikaa. Siksi viitteet muihin objekteihin on hyvä hakea pelin käynnistyessä kerran ja tallentaa ne muuttujiin, jolloin niitä voidaan käyttää ilman jatkuvaa hakemista. [14.]

Toinen, hieman ohjelmoijan työtä helpottava keino, on ”viestin” lähettäminen Unityn SendMessage-metodin avulla.

SendMessage on Unityn ominaisuus, jonka avulla voi lähettää peliobjektille ”viestinä” metodin nimen ja tätä metodia yritetään kutsua peliobjektin jokaisesta skripti-komponentista. SendMessagella on mahdollista lähettää kutsuttavalle metodille yksi parametri, mutta paluuarvoa ei ole mahdollista saada. [15.]



```
513     void OnLevelWasLoaded(int loadedLevel){
514
515         // Continue from where the music left off
516         musicManager.PlayFrom(currentMusicTime);
517
518         // Find the finish of the new level
519         finish = GameObject.FindGameObjectWithTag("Finish");
520
521         // Get the "par" time of the current level. It's set in the level's finish object.
522         timer.SetMaxTime(finish.GetComponent<Finish>().getParTime());
523
524         // Update the current level index
525         currentLevel = loadedLevel;
526
527         // Find all collectables in this level
528         collectables = GameObject.FindGameObjectsWithTag("Collectable");
529         numberOfCollectables = collectables.Length;
530
531         player.SendMessage("StartLevel", SendMessageOptions.DontRequireReceiver);
532
533         HideAllScreens();
534     }
```

Kuva 9. Esimerkki SendMessage-metodin käytöstä OnLevelWasLoaded-metodissa.

Kuvassa 9 on esitetty OnLevelWasLoaded-tapahtumametodi, jota kutsutaan aina kun Unityn LoadLevel-metodi on suoritettu loppuun. Metodissa käydään ensin läpi tason latauksen jälkeiset rutiinitoimenpiteet ja lopuksi lähetetään pelaajalle (player) viesti SendMessage-metodilla. DontRequireReceiver-parametrilla estetään virheilmoitus tilanteessa, jossa annetun nimistä metodia ei löydy yhdestäkään objektin skriptistä. SendMessage on kätevä erityisesti tilanteissa, jossa samannimistä metodia pitää kutsua useasta eri skriptistä samassa peliobjektissa.

Kolmas tapa on käyttää skriptissä staattisia muuttujia, jolloin muuttujien kutsumiseen ei tarvitse suoraa viittausta skriptin instanssiin, vaan muuttujiin päästään käsiksi pelkän skriptin nimen avulla. Staattiset muuttujat saavat jokaisessa skriptin instanssissa saman arvon. Niiden käyttö ei siis ole yleensä mielekästä tapauksissa, joissa samaa skriptiä tarvitaan useissa peliobjekteissa. [16.]

2.6.3 Editori-skriptit

Peliobjekti-skriptien lisäksi Unitylle on mahdollista kirjoittaa editoria ja sen toimintaa muokkaavia skriptejä. Tämä mahdollistaa erilaisten laajennusten, kuten uusia työkalujen luomisen. Skriptien kirjoittaminen ei vaadi mitään erikoista, vaan skriptin luokka periytetään vain MonoBehaviour-luokan sijaan jostakin laajentavasta luokasta, joita on kolme: Editor Window, Property Drawer ja Custom Editor. [17.]

Editor Window-luokan avulla luodaan uusi ikkuna Editoriin. Ohjelmoija voi itse päättää mitä työkaluja ja sisältöä ikkunan sisälle tulee. Tällaista ikkunaa voi esimerkiksi käyttää

tiedostojen tuomiseen projektiin. Property Drawer-luokkaa voi käyttää muuttamaan joidenkin kontrollien tai muuttujien ulkonäköä inspectorissa. Esimerkiksi jotkin luokat saattavat oletusarvoisesti näyttää sekavilta tai sotkuisilta inspectorissa, joten tällaisille saattaa olla mielekästä suunnitella oma Property Drawer. Custom Editorilla voidaan luoda mukautettuja editoreja komponenteille.

3 Kehitettävä peli

3.1 Pelin määrittely

Mobiilialustan kosketusnäyttökäyttöliittymä asettaa omat rajoituksensa pelimekaniikan toteutukseen. Pelin tulee olla yksinkertainen, mutta ei yksitoikkoinen. Päädyinkin peligenren valinnassa ongelmanratkaisutyyppiseen taitopeliratkaisuun. Kyseisen genren peleissä usein pelin taitoa tai harjoittelua vaativa osuus ei liity pelimekaniikkaan. Toisin sanoin peli on helppo oppia, mutta vaikea hallita. Tämä mahdollistaa suhteellisen runsaan ja mielekkään sisällön rakentamisen yksinkertaisen pelimekaniikan ympärille.

Pelityypin valitsemisen jälkeen tuli määritellä pelin tyyli ja sisältö, kuten se mitä pelissä tehdään ja mikä on pelin päämäärä. Asian selvittämiseksi kirjoitin pelisuunnitteludokumentin (engl. Game Design Document, GDD), jossa määrittelin pelimekaniikan, pelin tarkoituksen, käytetyt työkalut sekä peliin tyylin. Dokumenttiin ja määrittelyyn tuli tietenkin muutoksia kehitystyön edetessä.

3.2 Pelin idea

Pelin nimi on Orientia, joka on johdettu englannin kielen termistä ”orientation” eli suunta tai suuntautuminen. Pelin ideana on, että pelaajan tulee saada pelihahmona toimiva pallo kaksiulotteisen tason loppuun ennen ajan loppumista. Pelaaja ei pysty vaikuttamaan pallon liikkeisiin suoraan, vaan pallon liikuttaminen tapahtuu kameraa ja painovoiman suuntaa kääntämällä, mistä on myös peräisin pelin nimi. Pelissä on lisäksi keräiltäviä objekteja. Tason läpäiseminen vaatii, että pelaaja kerää kaikki tasosta löytyvät objektit. Vasta tämän jälkeen tason maali aukeaa.

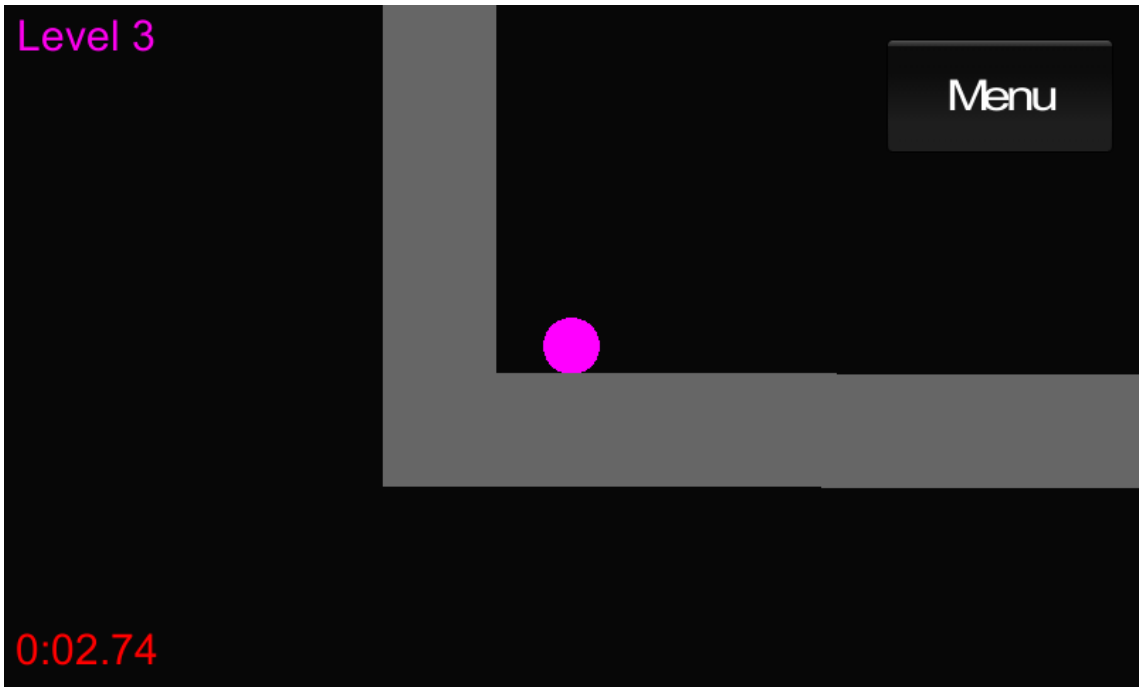
Pelissä on seitsemän tasoa, joista aluksi on auki vain yksi. Tasoja läpäisemällä pelaaja saa avattua lisää tasoja.

3.3 Pelin käyttöliittymä

Pelin käyttöliittymänä toimii kosketusnäyttö ja laitteen back-näppäin. Back-näppäintä käytetään ainoastaan, kun pelaaja haluaa palata eri valikkonäkymistä takaisin päävalikkoon. Muuten pelaajan ja pelin interaktio on täysin kosketusnäytön varassa. Pelissä ei ole ominaisuuksia, jotka tarvitsevat useaa samanaikaista kosketusta, vaan kaikki pelin toiminnot on tehtävissä yhdellä sormella.

3.4 Pelinäkymän graafinen käyttöliittymä

Varsinainen pelinäkymä on pyritty pitämään mahdollisimman yksinkertaisena. Pelin aikana ruudulla (kuva 10) näkyy varsinaisen peligrafiikan lisäksi ainoastaan kaksi tekstielementtiä ja yksi nappi.



Kuva 10. Kuvakaappaus pelistä. Kuvassa näkyvät kaksi tekstielementtiä sekä menu-nappi.

Tekstielementit kertovat kentän numeron sekä jäljellä olevan ajan, ja nappi pysäyttää pelin ja avaa näytölle taukoikkunan (kuva 11).

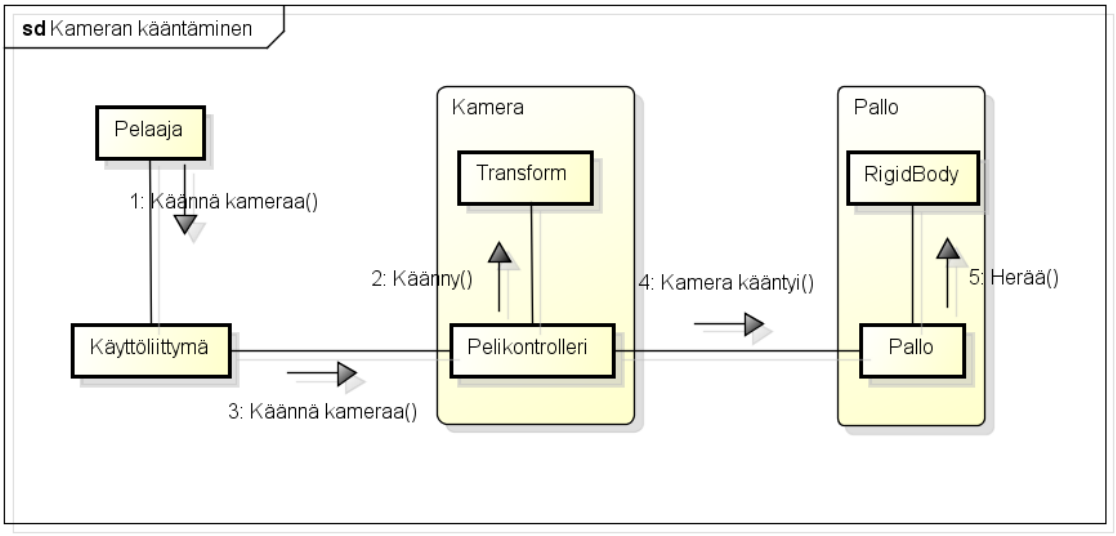
Kuva 11. Pelin taukotilassa näkyvä taukoikkuna.

Taukoikkunasta pelaaja voi joko jatkaa peliä, käynnistää tason uudelleen, siirtyä takaisin päävalikkoon tai lopettaa pelin.

3.5 Pelin rakenne

3.5.1 Pelikontrolleri

Pelin keskeisin osa on Game Controller -skripti, jonka kanssa kaikki muut pelin osat kommunikoivat. Game Controllerissa, eli pelikontrollerissa, on toteutettu kaikki pelin kulun hallinta, kuten valikoissa liikkuminen, tauko ja taukoikkunoiden hallinta sekä pelin mekaniikka, kuten kameran ja painovoiman kääntäminen (kuvio 1).



Kuva 12. Esimerkki pelin osien välisestä kommunikaatiosta

Turhien objekti- ja komponenttihakujen välttämiseksi pelikontrolleri-skripti on sijoitettu suoraan pelin kamera-objektiin, jolloin näillä on yhteinen transform-komponentti. Näin pelikontrollerin ei tarvitse hakea viittausta kameraan kääntääkseen sitä, vaan skriptissä voidaan käyttää suoraan sen omaa transform-komponenttia.

3.5.2 Valikot

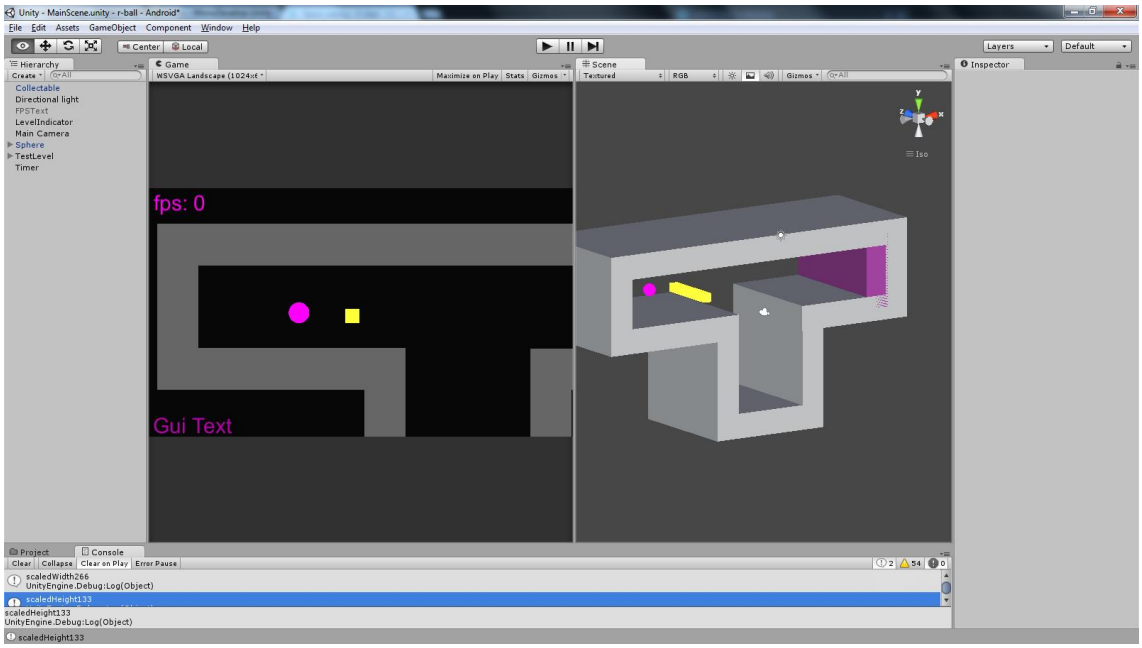
Jokainen valikkonäkymä on toteutettu omana skriptinään, jotka kaikki on liitetty kamera-objektiin. Skriptejä aktivoidaan ja deaktivoidaan yksi kerrallaan sen mukaan, minkä valikon tulee olla näkyvillä. Valikot luodaan kokonaan skriptien sisällä käyttäen Unityn GUI- ja GUILayout-luokkia.

3.6 Pelimaailman rakentaminen

Vaikka kyseessä onkin kaksiulotteinen peli, pelimaailma on kuitenkin mekaniikaltaan kolmiulotteinen. Pelihahmona toimiva pallo liikkuu suljetuissa kolmiulotteisissa kentissä kaksiulotteisesti.

3.6.1 Kentät ja pallo

Pelin kenttien rakentamiseen käytetään kolmiulotteisia palikoita. Sekä pallossa että palikoissa on Collision-komponentti, joka havaitsee objektien törmäykset toisiinsa ja estää niitä kulkemasta toistensä läpi. Pallossa on lisäksi RigidBody-komponentti, jonka ansiosta pallo toimii fysiikkamoottorin vaikutuksen alaisena.



Kuva 13. 3D-scene (oikealla) ja sen 2D-renderöinti (vasemmalla). Kuvassa näkyy myös keltainen keräiltävä objekti sekä violetti maalialue.

Kuvassa 13 nähdään, kuinka kolmiulotteiset objektit renderöidään kaksiulotteisiksi. Koska peli rakennetaan kolmiulotteisista objekteista, objektien liikettä tulee rajoittaa siten, että ne eivät liiku kolmannessa ulottuvuudessa. Näin saadaan simuloitua kaksiulotteista pelimaailmaa. Unityssä on mahdollista lukita RigidBody-komponentti eri tavoilla ja näin estää objektia pyörimästä tai kulkemasta tiettyyn suuntaan. Pallo-objekti on lukittu kulkemaan x-y -tasolla. Lukituksesta huolimatta objekti voi ”lipsahtaa” kentältä ulos esimerkiksi törmäyksen tai fysiikkamoottorin virhemarginaalin vuoksi. Tämän estämiseksi kentän seinät on tehty liioitellun syviksi. Näin varmistetaan, että pallo pysyy pelikentän sisällä.

3.6.2 Keräiltävät objektit

Pääkomponenttien lisäksi pelissä on keräiltäviä objekteja. Objektit ovat triggereitä eli peliobjekteja, joiden Collision-komponentti on määritelty triggeriksi. Triggerit havaitsevat törmäyksen toisiin Collision-komponentteihin, mutta läpipääsyn estämisen sijaan laukaisevat OnTriggerEnter-tapahtumametodin. Kun keräiltävä objekti havaitsee pallon osuvan siihen, se kutsuu tätä tapahtumametodia, jossa pelikontrollerille ilmoitetaan, että objekti on kerätty. Tämän jälkeen objekti tuhotaan, ja se poistuu pelistä.

Keräiltävät objektit on seinien tavoin tehty liioitellun syviksi. Näin varmistetaan, että pallo ei pääse missään tapauksessa kulkemaan objektin ohi, vaan törmäys havaitaan aina.

3.6.3 Pelin kamera

Kamera-objekti toimii pelin pääkontrollerina ja se sisältää suurimman osan pelin skripteistä. Kamera on määritelty ortogonaaliseksi ja sijoitettu siten, että se katsoo pelin tasoa ja pelihahmoa suoraan sivulta. Näin pelin kolmiulotteiset objektit renderöidään kaksiulotteisina ja niiden kolmiulotteisuus ei näy pelaajalle.

Kamerassa on Unityssä valmiina oleva SmoothFollow2D-skripti. Antamalla skriptille inspector-näkymässä viite johonkin peliobjektiin, saadaan kamera seuraamaan kyseistä peliobjektia pienellä viiveellä ja sulavasti.

3.7 Kameran ja painovoiman kääntäminen

Kameran ja painovoiman kääntäminen toteutetaan pelikontrolleriskriptissä, joka on kameraobjektin komponentti. Kamera-objektin transform-komponenttia käännetään 90 astetta myötä- tai vastapäivään sen mukaan, kumpaa puolta näytöstä pelaaja painaa. Painovoiman suunta määritellään kameran suunnan mukaisesti (kuva 14).

```
341     // Goes through on a fixed rate
342     void FixedUpdate() {
343
344     /*
345      * Actually, this is where the real "magic" happens.
346      * A simple example of trigonometry...
347      */
348
349         // Calculate the direction of the gravity from the rotation of the camera
350         Vector3 temp = transform.rotation.eulerAngles;
351         float tempX = Mathf.Sin(Mathf.Deg2Rad * temp.z);
352         float tempY = -Mathf.Cos(Mathf.Deg2Rad * temp.z);
353         temp.Set(tempX, tempY, 0);
354         temp.Normalize();
355         Physics.gravity = temp * gravity;
356     }
```

Kuva 14. Painovoiman kääntäminen kameran mukaan.

FixedUpdate-metodissa kameran transform-komponentin pyörimiskulmasta lasketaan painovoimalle suuntavektori käyttäen kaavoja $x = \sin(\alpha * y)$ ja $x = \cos(\alpha * y)$.

3.8 Pelin käynnistys ja kulku

Pelin sisältö on jaettu scene-tiedostoihin, joista tärkein on MainScene. MainScene sisältää pelin muuttumattomien osien, kuten pelihahmon ja kameran tiedot ja se ladataan pelin käynnistyessä Unityn LoadLevel-metodilla. LoadLevel-metodi poistaa kaikki peliin aikaisemmin ladatut scenet ja niiden peliobjektit, minkä vuoksi MainScenen kaikissa peliobjekteissa on skripti, jossa käytetään DontDestroyOnLoad-metodia. Tämä estää näiden objektien tuhoutumisen uuden kentän latauksen yhteydessä.

3.8.1 Tasojen lataaminen

Pelin seitsemän tasoa on jaettu siten, että jokainen taso on omassa scene-tiedostossaan. Kun jokin taso halutaan avata, tasoa vastaava scene-tiedosto ladataan LoadLevel-metodilla, jolloin jo mahdollisesti auki ollut aikaisempi taso tuhotaan.

Jokaiseen tasoon on määritetty Start-niminen peliobjekti. Tämä peliobjekti on näkymätön eikä sillä ole yhtään lisättyä komponenttia. Start-objektia tarkoitus on se, että sen koordinaatteja käytetään tason aloituskoordinaatteina. Kun tason scene-tiedosto ladataan LoadLevel-metodilla, pelin suoritus keskeytyy, kunnes scene on kokonaan ladattu. Tämän jälkeen, OnLevelWasLoaded-tapahtumametodissa, kutsutaan Pallo-objektin skriptistä StartLevel-metodia, jossa aloituskoordinaatit haetaan

Start-objektista ja pallo siirtyy näihin koordinaatteihin. Näin uusien tasojen rakentaminen helpottuu, sillä tason aloituspaikkaa voidaan muuttaa melko vapaasti. Haittavaikutuksena ratkaisulla on se, että pallo voidaan siirtää aloituskoordinaatteihin vasta, kun peli käynnistyy. Tämä aiheuttaa sen, että kamera ei ole oikeassa paikassa, kun peli alkaa, joten se joutuu siirtymään pelaajanseuraamisskriptinsä mukaisesti, mikä näkyy pelaajalle.

3.9 Pelin tallentaminen ja lataaminen

Tieto pelaajan edistymisestä tallennetaan pelistä poistuttaessa, joten tieto säilyy pelisessiosta toiseen. Pelikontrolleriskriptissä on määritelty levels-taulukko, joka sisältää kaikkien pelin tasojen avoimuustiedon eli tiedon siitä, onko taso auki vai ei. Kun pelaaja poistuu pelistä, taulukko tallennetaan Unityn tarjoamaan PlayerPrefs-luokkaan. PlayerPrefs-luokkaan ei voi tallentaa taulukoita suoraan, vaan taulukko joudutaan muuntamaan merkkijonoksi ennen tallentamista. Taulukko muunnetaan merkkijonoksi käyttäen BinaryFormatter- sekä MemoryStream-luokkia (kuva 14).

```
420     /// <summary>
421     /// Saves the game progress.
422     /// </summary>
423     public void SaveGame() {
424         //Make a temporary list of the array
425         List<Level> tempList = levels.ToList();
426         //Get a binary formatter
427         BinaryFormatter b = new BinaryFormatter();
428         //Create an in memory stream
429         MemoryStream m = new MemoryStream();
430         //Save the scores (using temporary list)
431         b.Serialize(m, tempList);
432         //Add it to player prefs
433         PlayerPrefs.SetString("Levels",
434             Convert.ToBase64String(
435                 m.GetBuffer()
436             )
437         );
438     }
```

Kuva 15. Pelin tallentaminen.

Peli ladataan toistamalla kuvan 15 prosessi käänteisessä järjestyksessä (kuva 16). Jos PlayerPrefs ei sisällä haluttuja tietoja, latausmetodi palauttaa arvon false. Tämä tarkoittaa usein sitä, että peli käynnistetään ensimmäisen kerran, joten pelin tasojen tiedot initialisoidaan.

```
    /// <summary>
    /// Loads the game. Returns true if load is successful.
    /// </summary>
    /// <returns>
    /// If load is successful.
    /// </returns>
    public bool LoadGame(){

        //Debug.Log("Loading game");
        //Get the data
        string data = PlayerPrefs.GetString("Levels");
        //If not blank then load it
        if(!string.IsNullOrEmpty(data))
        {
            //Binary formatter for loading back
            BinaryFormatter b = new BinaryFormatter();
            //Create a memory stream with the data
            MemoryStream m = new MemoryStream(Convert.FromBase64String(data));
            //Load back the scores
            List<Level> tempList = (List<Level>)b.Deserialize(m);
            levels = tempList.ToArray();
            // If for some reason the number of levels has changed make a new level array
            if(numberOfLevels+1 != levels.Length){
                // Make a temporary array
                Level[] tempLevels = new Level[numberOfLevels+1];
                // Put all the values from old array into new array
                for(int i = 1; i < levels.Length; i++){
                        tempLevels[i] = levels[i];
                }
                // Finally, assign the fresh array into the old variable
                levels = tempLevels;
            }
            return true;
        }
        else{
            return false;
        }
    }
```

Kuva 16. Pelin lataaminen.

Pelin lataamisen yhteydessä otetaan myös huomioon pelin sisältämien tasojen määrä tai siinä mahdollisesti tapahtunut muutos. Näin peliin voidaan lisätä tasoja ilman pelaajan tietojen häviämistä.

4 Google Play

Google Play, aikaisemmin Android Market, on Googlen kehittämä ja tarjoama market-palvelu Android-käyttöjärjestelmälle suunnattujen sovellusten sekä digitaalisen median jakamiseen ja myymiseen. Android on avoimen lähdekoodin käyttöjärjestelmä, joten sovelluksia voi julkaista periaatteessa kuka tahansa. Ilmaisten sovellusten julkaiseminen vaatii ainoastaan Google Play -kehttäjäkonsolitilin, jonka rekisteröiminen maksaa 25 dollaria. [18.]

4.1 Beta- ja alpha-testaus

Peli voidaan julkaista beta- tai alpha-testiin ennen varsinaiseen kauppaan julkaisemista. Tällöin käyttäjä määrittää julkaisun yhteydessä Google Community-sivun tai ryhmäsähköpostiosoitteen, joka kertoo sen, ketkä pääsevät peliin käsiksi. Testaajat voivat antaa palautetta suoraan palveluun, joten testaajien ja kehittäjien ei tarvitse keskenään olla yhteydessä.

4.2 Maksulliset sovellukset

Maksullisen sovelluksen julkaisuun tarvitaan Google Play -kehittäjätilin lisäksi Google Wallet -kauppiaskeskuksen tilin. Kaikki Google Play -palvelussa julkaistavat sovellukset voidaan julkaista maksullisina. Kerran jo ilmaisena julkaistua sovellusta ei kuitenkaan voi jälkeenpäin muuttaa maksulliseksi. Google perii Google Play -palvelussa myytävistä sovelluksista maksutapahtumamaksun, joka on suuruudeltaan 30 prosenttia tuotteen myyntihinnasta. Myyjälle maksettava osuus esimerkiksi 10 euron hintaisesta tuotteesta on siis seitsemän euroa. [19.]

4.3 Sovelluksen sisäinen laskutus

Sovelluksissa voidaan toteuttaa sovelluksen sisäistä laskutusta, jolloin käyttäjät voivat ostaa omistamaansa sovellukseen lisäsisältöä. Sovelluksen sisäistä laskutusta voidaan käyttää sekä maksullisissa, että maksuttomissa sovelluksissa. Sovellus voidaan siis julkaista ilmaisena, mutta sovelluksen sisällä voi esimerkiksi olla lukittua sisältöä, jota voidaan myydä käyttäjälle. Myös Google Playn sovelluksen sisäisiin maksutapahtumiin sovelletaan tavallista 30 prosentin maksutapahtumamaksua. [20, 21.]

5 Pohdinta

Syitä siihen, miksi päätin valita Unityn insinöörityöni aihealueeksi ja työkaluksi, on monia. Yksi on se, että Unity oli minulle jo ennestään tuttu. Toinen on Unityn Android-lisenssin siirtyminen Unityn maksuttoman lisenssin alle. Päätökseen vaikutti kuitenkin eniten se, että mielestäni Unity on mahtava.

Unity poikkeaa käyttöliittymältään ja mekaniikaltaan ainakin minulle aikaisemmin tutuista kehitysympäristöistä. Täytyy kuitenkin sanoa, että vertailu on omalta osaltani hieman vaikeaa, sillä Unity on ensimmäinen täysin pelinkehitykseen suunniteltu kehitysympäristö, jota olen käyttänyt.

Omalaatuisuudestaan huolimatta, tai ehkä juuri siitä johtuen, Unity on erittäin helposti lähestyttävissä. Unityn yhteisöllisyys, laaja käyttäjäkunta sekä kattava dokumentaatio, tekevät Unitystä erittäin aloittelijaystävällisen. Unityn omat tutoriaalit ja esimerkkiprojektit ovat pääosin yksityiskohtaisia ja selkeitä. Näitä täydentävät lukuisten Unityn käyttäjien tekemät opetusvideot ja -dokumentit.

5.1 Projektin toteutus Unityllä

Pelinkehitysprojekti oli mielenkiintoinen prosessi. Peli sai alkunsa keväällä 2013, jolloin harjoittelin Unityn käyttöä osana työharjoitteluani. Harjoittelun vuoksi tein pohjan pelimekaniikalle, josta myöhemmin muodostui Orientia eli tämän insinöörityön projekti. Suurin osa pelin kehitystyöstä tehtiin kuitenkin samana syksynä.

Unityä mainostetaan helppokäyttöisenä, mikä osoittautui projektin aikana pitävän paikkansa. Projektin aikana ei mittavia ongelmia esiintynyt, mutta pienempiä kylläkin. Vaikka Unity oli minulle ennestään tuttu, minulla oli vielä paljon opittavaa sen käytöstä. Suurin osa ongelmista johtuikin kokemuksen puutteesta.

Eniten ongelmia aiheutti Unityn normaalista poikkeava skriptien ajaminen. Vaikeuksia oli etenkin skriptien ajamisjärjestyksen ja objektien välisen kommunikoinnin kanssa. Unityn suuri ja mahtava käyttäjäyhteisö oli kuitenkin suurena apuna tässä. Etenkin Unityn keskustelufoorumi ja sen yhteydessä elävä Unity Answers -yhteisö pitivät huolen siitä, että epäselvyyksiin ja ongelmatilanteisiin löytyi ratkaisu nopeasti.

Suurin yksittäinen ongelma liittyi pelin datan tallentamiseen ja lataamiseen. Ongelmana oli saada pelin tasojen tiedot sisältävä taulukko tallennettua Unityn PlayerPrefs-luokkaan. Luokkaan ei kuitenkaan pystynyt tallentamaan taulukkomuotoista dataa. Ratkaisu löytyi jälleen muiden Unityn käyttäjien yhteisöllisyyden ansiosta, sillä eräiden Unityn käyttäjien ylläpitämä sivustolta löytyi ohjeet ongelman ratkaisemiseen. [22.]

Olin aluksi epävarma siitä, miten Unityn kolmiulotteisista kappaleista tuottama kaksiulotteinen grafiikka pyörii mobiililaitteissa. Eri laitteilla testattaessa ei kuitenkaan ilmeentynyt minkäänlaisia ongelmia tehonpuutteen tai grafiikan piirtämisen kanssa. Nykyaikaisilla mobiililaitteilla tämä ei ole tietenkään mikään yllätys, sillä kehitetty peli ei ole grafiikallisesti tai laskennallisesti raskaimmasta päästä. Unitystä löytyy myös mobiililaitteille optimoituja vaihtoehtoja monille komponenteille, kuten esimerkiksi varjostimille.

5.2 Unityn soveltuvuus pelinkehitykseen

5.2.1 Hyvät puolet

Unity on suunniteltu siten, että sitä voivat käyttää muutkin kuin ohjelmoijat. Ohjelmointitaidoista on toki hyötyä, mutta käyttäjän ei välttämättä tarvitse osata ohjelmoida tai ymmärtää ohjelmakoodia juuri ollenkaan, sillä Unity hoitaa itse matalan tason tehtävät peli-ikkunan avaamisesta aina fysiikanmallinnukseen asti. On periaatteessa täysin mahdollista luoda peli Unityllä kirjoittamatta riviäkään ohjelmakoodia käyttämällä ainoastaan valmiita skriptejä, joita muut Unityn käyttäjät ovat kirjoittaneet. Mitään mullistavaa tai uutta näin ei tietenkään voi tehdä, mutta kokonaisen yksinkertaisen pelin tekemiseen pystyy Unityllä kuka vain.

Unityn kauneus piileekin juuri siinä, että se avaa kaikille oven pelinkehityksen maailman. Samalla avautuu ovi mahdollisesti taitaville artisteille tai pelisuunnittelijoille, joilla ei ole aikaisemmin ollut mahdollisuutta tai resursseja pelien tekemiseen. Suunnittelijat voivat keskittyä paremmin sisällön luomiseen, kun heidän ei tarvitse tehdä ”likaista työtä”, kuten laskea kameran liikuttamiseen liittyvää matematiikkaa.

Yksi Unityn parhaista ominaisuuksista on sen monipuolisuus, erityisesti mahdollisuus usealle alustalle kehittämiseen. Eri alustoille kehittäminen on Unityssä lähes naurettavan helppoa. Tässäkin asiassa Unity tekee itse likaisen työn, ja kohdealustan vaihtaminen hoituu nappia painamalla. Kohdealustan vaihtaminen ei juuri vaadi muutoksia projektiin tai skripteihin. Käyttöliittymän erot tulee tietenkin huomioida, sillä esimerkiksi näppäimistölle suunniteltu ohjelma ei automaattisesti toimi kosketusnäytöllä. Lisäksi on joitakin tilanteita, joissa kohdealustan vaihtaminen aiheuttaa ongelmia. Tällaiset tilanteet ovat tosin harvinaisia. Periaatteessa peliä ei

tarvitse tehdä uudestaan aina alustaa vaihdettaessa, vaan yhden projektin voi hetkessä julkaista usealle alustalle.

5.2.2 Huonot puolet

Ohjelmoijan näkökulmasta Unityn helppokäyttöisyys ja omalaatuinen lähestymistapa koodin ja skriptien ajamiseen eivät ole vain hyviä asioita. Unity pyrkii hoitamaan kaikki matalan tason tehtävät itse. Tämä on hyvä asia niille Unityn käyttäjille, joiden vahvuuksiin ohjelmointi ei lukeudu. Ohjelmointitaitoiset käyttäjät saattavat kuitenkin turhautua Unityn automatisoituihin ominaisuuksiin, sillä ne tekevät pelien optimoinnin vaikeaksi, tai jopa mahdottomaksi.

Unityn ilmaisversion ylivoimaisesti suurin heikkous maksulliseen verrattuna on sen huono ryhmätyöskentelyn tukeminen. Ilmainen versio sopii erittäin hyvin henkilökohtaisiin projekteihin, mutta jo kahden henkilön kokoinen tiimi tulee varmasti kokemaan useamman konfliktipainajaisen projektin versionhallinnassa. Ainakin Unity Pro -lisenssin hankkiminen on käytännössä pakollista, mikäli Unityllä haluaa työskennellä useamman kuin kahden käden voimin. Kehitystiimi voi harkita Team License -lisenssin hankkimista.

Team License -lisenssin käyttäminen on kallis investointi. Lisenssi maksaa 500 dollaria, mikä tuntuu kohtuulliselta, mutta lisenssi on oltava kehitystiimin jokaisella jäsenellä. Lisäksi lisenssin hankkiminen edellyttää, että käyttäjällä on jo ennestään Unity Pro -lisenssi, joka maksaa 1 500 dollaria. Esimerkiksi kolmen hengen kehitystiimi joutuu täydestä ryhmätyötuesta maksamaan kolmesta Unity Pro -lisenssistä ja kolmesta Team License -lisenssistä yhteensä 6000 dollaria. Käyttäjien on tietenkin mahdollista käyttää ulkoista versionhallintaohjelmistoa, mutta tämäkin vaatii vähintään Unity Pro -lisenssin kaikilta ryhmän jäseniltä, jotta he voisivat työskennellä projektissa samanaikaisesti ilman ongelmia.

Unity tuntuu mainiolta kehitystyökalulta indie-pelinkehittäjille ilmaislisenssinsä vuoksi. Unity Pro -lisenssin 1 500 dollarin hintakaan ei tunnu vielä niin suurelta investoinnilta, kun ottaa huomioon, miten paljon ominaisuuksia sillä saa. Pelkällä Unity Pro -lisenssillä on kuitenkin mahdollista käyttää näitä ominaisuuksia vain niille alustoille, jotka kuuluvat Unity Pro –lisenssiin.. Pro-tason ominaisuuksien käyttöönotto muilla alustoilla vaatii alustakohtaisen Pro-lisenssin, josta veloitetaan taas 1 500 dollaria. Jos esimerkiksi

indie-kehittäjä haluaa kehittää Unity Pro -tasoisia pelejä Androidille, hänen tulee ostaa sekä Unity Pro - että Android Pro -lisenssi. Näin vain yhdelle alustalle julkaiseminen maksaa 3 000 dollaria, tai mikäli tiimissä on esimerkiksi kolme jäsentä, 9 000 dollaria. Mikäli tiimi haluaa julkaista pelinsä myös iOS-alustalle, mikä on melko yleistä, jokaisen tiimin jäsenen tulee ostaa vielä iOS Pro -lisenssi, joka on myös 1 500 dollarin hintainen. Mielestäni tällainen hinnoittelu on hieman omituinen, sillä jokainen tiimin jäsen maksaa käytännössä 1 500 dollaria mahdollisuudesta kehittää peli tietokoneille, mistä tiimi tuskin hyötyy. Hinnoittelu olisi ehkä järkevämpi, jos alustakohtaisille Pro-lisenssien ostamiseen ei vaadittaisi myös Unity Pro -lisenssiä.

Lähteet

1 Andrews, Keith. 2013. UPDATE: Unity goes free on Android and iOS. Verkkodokumentti. <http://www.pocketgamer.biz/r/PG.Biz/Unity+news/news.asp?c=51030>. 21.5.2013. Luettu 7.11.2013.

2 Unity Technologies Celebrates 5 Years of Unity. Verkkodokumentti. <http://unity3d.com/http%3A//unity3d.com/company/public-relations/news/unity-5year-press>. 7.7.2010. Luettu 14.11.2013.

3 Sunsetting Flash. Verkkodokumentti. <http://blogs.unity3d.com/2013/04/23/sunsetting-flash/>. 23.4.2013. Luettu 15.11.2013.

4 License Comparisons. Verkkodokumentti. <http://unity3d.com/unity/licenses>. Luettu 14.11.2013.

5 Pricing. Verkkodokumentti. <https://store.unity3d.com/products/pricing>

6 Helgason, David. 2013. Putting the power of Unity in the hands of every mobile developer. Verkkodokumentti. <http://blogs.unity3d.com/2013/05/21/putting-the-power-of-unity-in-the-hands-of-every-mobile-developer/>. 21.5.2013. Luettu 15.11.2013.

7 Play well with others. Verkkodokumentti. <http://unity3d.com/unity/collaboration>. Luettu 15.11.2013.

8 Using External Version Control Systems with Unity. Verkkodokumentti. <http://docs.unity3d.com/Documentation/Manual/ExternalVersionControlSystemSupport.html>. 15.5.2013. Luettu 15.11.2013.

9 Verkkotietojärjestelmä. <http://docs.unity3d.com/Documentation/Manual/TextualSceneFormat.html>. 18.9.2013. Luettu 15.11.2013.

10 Verkkotietojärjestelmä. <http://en.wikipedia.org/wiki/Unity_%28game_engine%29>. 13.11.2013. Luettu 15.11.2013.

11 Unity physics. Verkkodokumentti. <http://unity3d.com/unity/quality/physics>. Luettu 15.11.2013.

12 Unity Technologies Launches 3rd Party Marketplace 'Unity Asset Store'. Verkkodokumentti. <http://www.marketwired.com/press-release/Unity-

Technologies-Launches-3rd-Party-Marketplace-Unity-Asset-Store-1350977.htm>. 10.11.2010. Luettu 15.11.2013.

13 Execution Order of Event Functions. Verkkodokumentti. <http://docs.unity3d.com/Documentation/Manual/ExecutionOrder.html>. 3.5.2013. Luettu 15.11.2013.

14 Verkkodokumentti. <http://docs.unity3d.com/Documentation/ScriptReference/GameObject.Find.html>. Luettu 15.11.2013.

15 Verkkodokumentti. <http://docs.unity3d.com/Documentation/ScriptReference/GameObject.SendMessage.html>. Luettu 15.11.2013.

16 DimasTheDriver. 2011. Using static variables in Unity3D. <http://www.41post.com/3424/programming/using-static-variables-in-unity3d>. 16.3.2011. Luettu 15.11.2013.

17 Extending the Editor. Verkkodokumentti. <http://docs.unity3d.com/Documentation/Components/ExtendingTheEditor.html>. 15.8.2013. Luettu 15.11.2013.

18 Kehittäjäksi rekisteröityminen. Verkkodokumentti. <https://support.google.com/googleplay/android-developer/answer/113468>. Luettu 19.11.2013.

19 Tapahtumakulut. Verkkodokumentti. <https://support.google.com/googleplay/android-developer/answer/112622?hl=fi>. Luettu 19.11.2013.

20 Sovelluksen sisäinen laskutus: esittely. Verkkodokumentti. <https://support.google.com/googleplay/android-developer/answer/1153479>. Luettu 19.11.2013.

21 Sovelluksen sisäinen laskutus: saatavuus ja käytännöt. Verkkodokumentti. <https://support.google.com/googleplay/android-developer/answer/1153481>. Luettu 19.11.2013.

22 Saving Data #1 – Remember Me?. Verkkodokumentti. <http://unitygems.com/saving-data-1-remember-me/>. 14.11.2012. Luettu 22.10.2013.

Game Design Document

Game Design Document

Orientia

Table of Contents

1 Game Overview

1.1 Concept

Orientia is a puzzle/skill game for mobile devices. The object of the game is to navigate through a 2D level by controlling the direction of the gravity in the game.

1.2 Target Audience

Orientia is basically fit for all ages. However, due to the complex nature of the game mechanics the main target audience will be teenagers and adults.

1.3 Tools

All software used in creating the game and its contents should be free.

1.3.1 Development Tools

The game will be designed and produced with Unity. The scripts used in the game will be written mainly in C# using the MonoDevelop IDE.

1.3.2 Other Tools

Artwork will be created using Inkscape, an open source vector graphics editor. All music used in the game are created or will be created with AudioSauna, a free music software.

2 Gameplay

2.1 Goal of the Game

In order to finish a level, the player – a ball – must navigate through a level and reach the finish zone. However, the finish zone must be opened before the player can enter it. In order to do this, the player must collect all collectibles from the level.

2.2 Controls

The player will only have the power to turn the direction of the gravity around. No direct interaction with the character (the ball) will be possible.

3 Graphics

3.1 Look and Feel

This game will be going for a ”futuristic” look. The color scheme of the game will be dark, the two main colors being black and purple. Overall, the game will be simplistic in design using only basic shapes. (such as cubes, spheres, etc.)

3.2 Art

The game will have little artwork, apart from the game logos. The game uses only basic shapes and the menus are mostly text, although some default graphics provided by Unity will be used (such as buttons).

3.3 Rendering

The game will be two-dimensional to minimize possible physical effects (such as nausea) to the player caused by the turning camera. The game is designed with three-dimensional objects that are rendered as two-dimensional by using an ortographic camera.

4 User Interface

4.1 In Game

The game is controlled via a touch screen. The controls are simple: The player turns the camera clockwise and counter-clockwise by touching the appropriate side of the game screen (Right = clockwise; Left = Counter-Clockwise). Turning the camera also turns the gravitational force so that it always points straight down.

4.2 Menus

Menu screens are navigated by using the touch screen and the back button of the device.

5 Music and Sounds

The game will have no sound effects, although it will feature background music in the main menu and in the levels.